JSP - SECURITY

http://www.tutorialspoint.com/jsp/jsp_security.htm

JavaServer Pages and servlets make several mechanisms available to Web developers to secure applications. Resources are protected declaratively by identifying them in the application deployment descriptor and assigning a role to them.

Several levels of authentication are available, ranging from basic authentication using identifiers and passwords to sophisticated authentication using certificates.

Role Based Authentication:

The authentication mechanism in the servlet specification uses a technique called role-based security. The idea is that rather than restricting resources at the user level, you create roles and restrict the resources by role.

You can define different roles in file tomcat-users.xml, which is located off of Tomcat's home directory in conf. An example of this file is shown below:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
<role rolename="tomcat"/>
<role rolename="role1"/>
<role rolename="manager"/>
<role rolename="admin"/>
<user username="tomcat" password="tomcat" roles="tomcat"/>
<user username="role1" password="tomcat" roles="role1"/>
<user username="both" password="tomcat" roles="tomcat,role1"/>
<user username="admin" password="secret" roles="admin,manager"/>
</tomcat-users>
```

This file defines a simple mapping between user name, password, and role. Notice that a given user may have multiple roles, for example, user name="both" is in the "tomcat" role and the "role1" role.

Once you identified and defined different roles, a role-based security restrictions can be placed on different Web Application resources by using the **<security-constraint>** element in web.xml file available in WEB-INF directory.

Following is a sample entry in web.xml:

```
<web-app>
...
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>
               SecuredBookSite
            </web-resource-name>
            <url-pattern>/secured/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
            <description>
            Let only managers use this app
            </description>
            <role-name>manager</role-name>
        </auth-constraint>
    </security-constraint>
    <security-role>
      <role-name>manager</role-name>
    </security-role>
    <login-config>
      <auth-method>BASIC</auth-method>
    </login-config>
...
```

```
</web-app>
```

Above entries would mean:

- Any HTTP GET or POST request to a URL matched by /secured/* would be subject to the security restriction.
- A person with manager role is given access to the secured resources.
- Last, the login-config element is used to describe the BASIC form of authentication.

Now if you try browsing to any URL including the /security directory, it would display a dialogue box asking for user name and password. If you provide a user "admin" and password "secrer" then only you would have access on URL matched by /secured/* because above we have defined user admin with manager role who is allowed to access this resource.

Form Based Authentication:

When you use the FORM authentication method, you must supply a login form to prompt the user for a username and password. Following is a simple code of login.jsp to create a form for the same purpose:

```
<html>
<body bgcolor="#ffffff">
   <form method="POST" action="j_security_check">
      <table border="0">
      <tr>
      <td>Login</td>
      <td><input type="text" name="j_username"></td>
      </tr>
      <tr>
      <td>Password</td>
      <td><input type="password" name="j_password"></td>
      </tr>
      </table>
      <input type="submit" value="Login!">
      </center>
   </form>
</body>
</html>
```

Here you have to make sure that the login form must contain form elements named j_username and j_password. The action in the <form> tag must be j_security_check. POST must be used as the form method. Same time you would have to modify <login-config> tag to specify auth-method as FORM:

```
<web-app>
...
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>
               SecuredBookSite
            </web-resource-name>
            <url-pattern>/secured/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
            <description>
            Let only managers use this app
            </description>
            <role-name>manager</role-name>
        </auth-constraint>
    </security-constraint>
    <security-role>
      <role-name>manager</role-name>
    </security-role>
    <login-config>
      <auth-method>FORM</auth-method>
      <form-login-config>
        <form-login-page>/login.jsp</form-login-page>
        <form-error-page>/error.jsp</form-error-page>
      </form-login-config>
    </login-config>
...
</web-app>
```

Now when you try to access any resource with URL /secured/*, it would display above form asking for user id and password. When the container sees the "j_security_check" action, it uses some internal mechanism to authenticate the caller.

If the login succeeds and the caller is authorized to access the secured resource, then the container uses a session-id to identify a login session for the caller from that point on. The container maintains the login session with a cookie containing the session-id. The server sends the cookie back to the client, and as long as the caller presents this cookie with subsequent requests, then the container will know who the caller is.

If the login fails, then the server sends back the page identified by the form-error-page setting

Here j_security_check is the action that applications using form based login have to specify for the login form. In the same form you should also have a text input control called j_username and a password input control called j_password. When you see this it means that the information contained in the form will be submitted to the server, which will check name and password. How this is done is server specific.

Check Standard Realm Implementations to understand how j_security_check works for Tomcat container.

Programmatic Security in a Servlet/JSP:

The HttpServletRequest object provides the following methods, which can be used to mine security information at runtime:

SN	Method and Description
1	**String getAuthType()** The getAuthType() method returns a String object that represents the name of the authentication scheme used to protect the Servlet.
2	**boolean isUserInRole(java.lang.String role)** The isUserInRole() method returns a boolean value: true if the user is in the given role or false if they are not.
3	**String getProtocol()** The getProtocol() method returns a String object representing the protocol that was used to send the request. This value can be checked to determine if a secure protocol was used.
4	**boolean isSecure()** The isSecure() method returns a boolean value representing if the request was made using HTTPS. A value of true means it was and the connection is secure. A value of false means the request was not.
5	**Principle getUserPrinciple()** The getUserPrinciple() method returns a java.security.Principle object that contains the name of the current authenticated user.

For example, a JavaServer Page that links to pages for managers, you might have the following code:

```
<% if (request.isUserInRole("manager")) { %>
<a href="managers/mgrreport.jsp">Manager Report</a>
<a href="managers/personnel.jsp">Personnel Records</a>
<% } %>
```

By checking the user's role in a JSP or servlet, you can customize the Web page to show the user only the items she can access. If you need the user's name as it was entered in the authentication form, you can call getRemoteUser method in the request object.